Aggregation-Based Gossip for Certificate Transparency

Rasmus Dahlberg
Karlstad University
rasmus.dahlberg@kau.se

Tobias Pulls
Karlstad University
tobias.pulls@kau.se

Jonathan Vestin
Karlstad University
jonathan.vestin@kau.se

Toke Høiland-Jørgensen
Karlstad University
toke.hoiland-jorgensen@kau.se

Andreas Kassler
Karlstad University
andreas.kassler@kau.se

ABSTRACT

Certificate Transparency (CT) is a project that mandates public logging of TLS certificates issued by certificate authorities. While a CT log is designed to be trustless, it relies on the assumption that every client sees and cryptographically verifies *the same log*. The solution to this problem is a gossip mechanism that ensures that clients share the same view of the logs. Despite CT being added to Google Chrome, no gossip mechanism is pending wide deployment. We suggest an aggregation-based gossip mechanism that passively observes cryptographic material that CT logs emit in plaintext, aggregating at packet processors and periodically verifying log consistency off-path. Based on 20 days of RIPE Atlas measurements that represents clients from 3500 autonomous systems and 40% of the IPv4 space, our proposal can be deployed incrementally for a realistic threat model with significant protection against undetected log misbehavior. We also discuss how to instantiate aggregation-based gossip on a variety of packet processors, and show that our P4 and XDP proof-of-concepts implementations run at line-speed.

1 INTRODUCTION

Usage of Transport Layer Security (TLS) on the web is gradually transitioning towards a security-by-default model. For example, 30% of Google's egress traffic uses an always encrypted protocol named QUIC [32], more and more websites adopt HTTP over TLS (HTTPS) [23], and Google Chrome is about to change its security indicators to better reflect that the status quo is no longer based on an opt-in model[1]. The ecosystem surrounding transport security has also undergone a paradigm shift. This includes the standardization of TLS 1.3[2], as well as automated, free-of-charge, and transparent certificate issuance with fewer errors [31, 37]. Given that the security of TLS is ultimately underpinned by certificates—identity to key bindings that trusted Certificate Authorities (CAs) issue—the latter is of particular importance. Automated and free certificate issuance is provided by today's largest and non-profit CA *Let's Encrypt*. Greater transparency is due to Certificate Transparency (CT) logging [34, 35], which is being deployed incrementally in Google Chrome: as of May 2018 new certificates must be publicly logged to be trusted[3], augmenting the weakest-link model of the CA ecosystem such that there is little or no blind trust left.

While the new requirement of including certificates into CT logs is a significant improvement, it is not without shortcomings. The design of CT is such that a log operator need not to be yet another trusted third-party. However, this requires mechanisms that either deter or prevent a log from serving different conflicting versions of its structure and content to the parties interacting with the log; so called *split views* [12, 41]. Proposals for these mechanisms often take the following retroactive form: log clients *gossip* signed material that the logs generate, thereby making it possible to challenge any log to prove that it is serving consistent views by leveraging the cryptographic foundation of CT. Gossip mechanisms are complex for a number of reasons that range from client privacy to varying threat models and deployment challenges [12, 18, 41]. At the time of writing and to the best of our knowledge, no gossip mechanism is widely used[4]. As such, clients must trust that CT logs refrain from presenting split views, and the community have yet to settle for one or more gossip mechanisms that can be deployed at scale.

We propose a gossip mechanism that assists in split view detection retroactively based around the idea of network packet processors–such as switches, routers, middleboxes, and operating systems—that *aggregate* signed log material in plaintext which is then used to challenge the logs to prove consistency off-path. This proposal is controversial given current trends to encrypt transport protocols, which is otherwise an approach that combats inspection of network traffic and protocol ossification [20, 30]. Nevertheless, the idea is similar to the debate of keeping parts of the multi-path QUIC header accessible to middleboxes for the sake of traffic shaping [13], whereas we argue that keeping gossip related material in plaintext comes with few downsides: encryption is often motivated by security considerations, but in our case it has the opposite effect and *reduces* the security of CT. Our gossip mechanism has no major negative impact on privacy, makes split views significantly more risky in a realistic threat model, is easy to implement, and would offer significant protection for a large fraction of the Internet with relatively small deployment. The three main limitations are (i) no protection against isolated clients which is beyond the scope of any retroactive gossip mechanism [49], (ii) reliance on clients that fetch easy-to-process cryptographic material from the logs in plaintext, and (iii) possible concerns surrounding protocol ossification [30].

[1] https://web.archive.org/web/20180519224524/https://blog.chromium.org/2018/05/evolving-chromes-security-indicators.html (May 2018)
[2] https://web.archive.org/web/20180519224403/https://www.ietf.org/mail-archive/web/ietf-announce/current/msg17592.html (March 2018)
[3] https://groups.google.com/a/chromium.org/forum/#!msg/ct-policy/wHILiYf31DE/iMFmpMEkAQAJ (February 2018), accessed 2018-05-20.

[4] Google provides a gossip package that supports 'minimal gossip' and the mechanisms of Nordberg *et al.* [41]: https://github.com/google/certificate-transparency-go/tree/master/gossip (May 2018). According to Gasser *et al.* up to 16,800 domains *might* support server-side gossip, but yet it appears that there is "next to no deployment in the wild" [24]. At least two monitors (Graham Edgecombe and SSLMate) support and receive gossip from the CT honey bee project [3], which exists as a stand-alone client daemon, a Google Chrome plugin, and an Android application named 'transparensbee'.

Notably we build upon Dahlberg's Master's thesis which explored the idea of aggregation, different plaintext sources, and the use of P4 for instantiation [43]. Specifically, our contributions are:

- Design and associated security considerations for a gossip mechanism based on passive aggregation of cryptographic material generated by transparency logs like CT and actively challenging logs off-path to prove consistency (Section 3).
- Generic implementations of passive aggregation in P4 [7] and XDP [25] for CT-over-DNS traffic, supporting programmable packet processors that range from Linux-based operating systems to network interface cards and routers (Section 4).
- A tailored P4 implementation for an Xilinx NetFPGA SUME board, which is used in a performance evaluation showing that aggregation can take place inline at 10 Gbps without any clear throughput-related distinguisher (Section 4).
- A simulation based on RIPE Atlas network measurements which uses 4606 world-wide probes over 20 days to evaluate the impact of deploying our gossip mechanism at Internet Autonomous Systems (ASes) and Internet Exchange Points (IXPs). Our evaluation shows that incremental roll-out of aggregation-based gossip at well-connected ASes and/or IXPs would protect a significant portion of all Internet clients from otherwise undetected split view attacks (Section 5).

Besides the sections referenced above, the paper is structured as follows. Section 2 provides necessary background on CT and programmable data planes. Related work is presented in Section 6, followed by discussion in Section 7 and conclusions in Section 8.

2 BACKGROUND

Throughout this paper the reader needs to be familiar with CT and the principles of programmable packet processors. Therefore, we start by motivating and describing each building block separately.

2.1 Certificate Transparency

The CA ecosystem is historically known for its weakest-link security: if one trusted third-party gets the certificate issuance process wrong, then a fraudulent identity-to-key binding can be issued for any domain [19]. Although multiple cases of certificate mis-issuance have been found in the past[5], it could be the tip of the iceberg because it is hard to determine what has been issued for whom. This dilemma is the motivation of CT [34, 35]. The idea is simple: TLS clients require that presented certificates must be disclosed in a public append-only tamper-evident log [14, 21], such that anyone can monitor the set of issued certificates. Notably the goal of CT is not to prevent certificate mis-issuance, but to *detect* it.

2.1.1 Building Blocks. Due to the underlying structure of a CT log, it is a cryptographically verifiable append-only tamper-evident data structure. At any given time, a signed snapshot can be generated that represents the structure and the content of the log. In CT jargon, this is called a Signed Tree Head (STH). It is possible to prove certificate inclusion by revealing a logarithmic number of hashes which are used to reconstruct the tree head of an STH. Upon match, this proves membership given an existentially unforgeable signature scheme and a collision resistant hash function [18, 40]. It is also possible to prove efficiently that two STHs are consistent, i.e., the log is append-only without tampering. This means that a client can verify whether the presented certificates are part of the log without fully downloading it, and whatever was in the log yesterday must still be there today. Unlike the CA ecosystem, such a setup requires no trusted party since correctness can be verified.

While the cryptographic foundation of CT is well-understood, it is not without deployment challenges. For example, there may be a halt in the certificate issuance process if a CA must wait for log inclusion. Therefore, a log can issue a *promise* to include: within some Maximum Merge Delay (MMD), the corresponding certificate must be appended to the log. In CT jargon this is called a Signed Certificate Timestamp (SCT), and it introduces an additional log component that must be audited. Today's deployment of CT evolves around a policy where Chrome clients check that certificates are accompanied by at least two SCTs[6]. Apple announced that a similar policy will be used[7], and perhaps this is also the case for Mozilla Firefox[8]. The status quo is thus to *trust* the logs, but as CT is being rolled out incrementally it would be a natural next step to *verify* that this trust is not misplaced by interacting with the logs [48].

2.1.2 Privacy. Suppose that a TLS client does challenge a log to prove certificate inclusion. In the same way that a revocation check leaks a client's browsing history to the CA, this would leak a client's browsing history to the log. Now suppose that a proof and associated STH are instead stapled by the TLS server: a similar dilemma arises if the STH is rarely served by other servers *and* if the client verifies that it is consistent with a current view. In other words, the process of auditing a CT log can result in privacy concerns. Two promising approaches that deal with these issues include (i) proxying all log interactions via a third-party that already knows a client's browsing history, and (ii) adding STH frequency restrictions which ensure that a d-day old STH is never rare. The former exists in the form of CT-over-DNS [33] (further described in Section 4.1) and the latter is discussed within IETF [34, 35, 41]. Another approach is to use private information retrievals [36].

2.1.3 Gossip. Despite the ability to cryptographically verify that a log includes certificates in append-only order, it does not mean much unless everybody observes the same log. A log that presents two different versions of itself is said to perform a *partitioning attack*, and the result is a *split view*. To see why this is a problem, suppose that a log serves a split view to a TLS client and a monitor. The client sees a consistent version of the log that includes a fraudulent certificate. Similarly, the monitor sees a consistent (but different) version of the log that excludes the fraudulent certificate. The client cannot distinguish between a benign and a fraudulent certificate, and the monitor does not see it in the first place. As such, the fraudulent certificate goes unnoticed, and CT fails to achieve its goal of detecting certificate mis-issuance because the log is cheating.

[5] https://web.archive.org/web/20180527220047/https://www.enisa.europa.eu/publications/info-notes/certificate-authorities-the-weak-link-of-internet-security (September 2016)

[6] https://github.com/chromium/ct-policy (May 2018)

[7] https://web.archive.org/web/20180605133051/https://support.apple.com/en-us/HT205280 (June 2018)

[8] https://docs.google.com/document/d/1rnqYYwscAx8WhS-MCdTiNzYQus9e37HuVyafQvEeNro/edit (Draft 0.1.0), accessed 2018-06-12

The theoretic assumption that CT relies on is a perfect gossip mechanism: as soon as a log makes a statement, it is immediately visible to public scrutiny. For example, in the example above the monitor would first observe an STH from the client's split view and then challenge the log to prove consistency (which it cannot). While CT gossip is necessary to untrust the log and avoid creating a new form of 'trusted CA', it is challenging in practise due to the complexity of the setting which involves privacy concerns, legacy yet vital Internet infrastructure, and the CA ecosystem to name a few aspects [12, 41]. It should also be noted that the term *gossip* is not strict in the traditional[9] sense throughout the paper—we view CT gossip as any mechanism that prevents or deters a log from presenting split-views. The discussion of our work in relation to earlier approaches towards CT gossip is deferred until Section 6.

2.2 Data Plane Programmability

Equipment such as switches, routers, and Network Interface Cards (NICs) are typically integrated tightly using customized hardware and Application-Specific Integrated Circuits (ASICs). This inflexible design limits the potential for innovation and leads to long product upgrade cycles, where it takes *years* to introduce new processing capabilities and support for different protocols and header fields (mostly following lengthy standardization cycles). The recent shift towards flexible *match+action* packet-processing pipelines—including RMT [8], Intel Flexpipe [44], Cavium XPA[10], and Barefoot Tofino[11]—have the potential to change the way in which packet processing hardware is implemented: it enables programmability using high-level languages such as P4 (see below), while at the same time maintaining performance comparable to fixed-function chips.

2.2.1 P4. The main goal of P4 is to simplify the programming of protocol-independent packet processors by providing an abstract programming model for the network data plane [7]. In this setting the functionality of a packet processing device is specified without assuming any hardwired protocols and headers. Consequently, a P4 program must parse headers and connect the values of those protocol fields to the actions that should be executed based on a pipeline of reconfigurable match+action tables. Additionally, per packet metadata fields can be used for the processing and state management, augmented by customizable registers, meters, and counters. The functionality of the core language is quite limited and traditional building blocks such as for-loops, recursion, and floating point operations are undefined. Some of these limitations can be removed, but at the cost of platform dependencies that connect P4 with external functions which may be more complex.

Once a P4 program is specified, a front-end compiler generates a high-level intermediate representation that a back-end compiler uses to create a target-dependent program representation. Compilers are already available for several platforms, including the software-based simple switch architecture[12] (also called the behavioral model), SDNet for Xilinx NetFPGA boards [10], and Netronome's smart NICs which are capable of running external functions in sandboxed C-environments [45]. There is also support for compiling basic P4 programs into eBPF byte code[13].

2.2.2 XDP. The Berkeley Packet Filter (BPF) is a Linux-based packet filtering mechanism [38]. Verified bytecode is injected from user space, and executed for each received packet in kernel space by a just-in-time compiler. Extended BPF (eBPF) enhances the original concept, enabling faster runtime and many new features[14]. For example, an eBPF program can be attached to the Linux traffic control tool `tc`, and additional hooks were defined for a faster eXpress Data Path (XDP) [25]. In contrast to the Intel Data Plane Development Kit (DPDK) which runs in user space and completely controls a given network interface supporting a DPDK driver[15], XDP cooperates with the Linux stack to achieve fast, programmable, and reconfigurable packet processing.

3 AGGREGATION-BASED GOSSIP

An overview of aggregation-based gossip is shown in Figure 1. The setting (of which there will be multiple instances) consists of a log that sends plaintext STHs to a client over a network, and as part of the network an inline *packet processor* passively aggregates observed STHs to an off-path *challenger* that challenges the log to prove consistency. A log cannot present split views to different clients that share an aggregating vantage point because it would trivially be detected by that vantage point's challenger. A log also cannot present split views to different challengers because they are off-path in the sense that they are indistinguishable from one another (e.g., using an anonymity network). This means that every client that is covered by an aggregator must be on the same view, i.e., otherwise a challenger detects an inconsistency and tells the world about it. Further, a client that is not covered by an aggregator can receive indirect protection in the form of herd immunity as discussed in Section 7.4. After introducing our threat model and main security notion, we describe the concept of the two components in greater detail: in-line STH aggregation and off-path challenging.

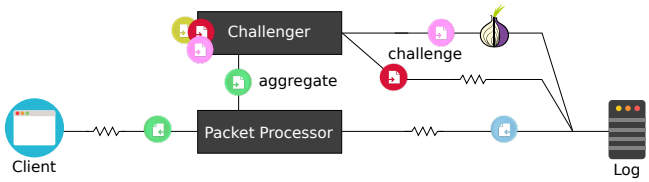


Figure 1: Our setting involves a Log that sends STHs to a Client in plaintext. These STHs are processed by a Packet Processor component that filters the traffic for STHs, aggregating to a Challenger component. The challenger challenges the log to prove consistency with regards to the aggregated STHs off-path, e.g., using an anonymity network.

[9] The traditional gossip problem involves n entities, each of which propose a unique message that must be propagated to every other entity in the group [6, 27].
[10] https://web.archive.org/web/20170707175037/https://cavium.com/newsevents-cavium-and-xpliant-introduce-a-fully-programmable-switch-silicon-family.html (n.d.)
[11] https://web.archive.org/web/20180105002028/https://barefootnetworks.com/products/brief-tofino/ (n.d.)
[12] https://github.com/p4lang/p4c-bm (April 2018)
[13] https://github.com/iovisor/bcc/tree/master/src/cc/frontends/p4 (May 2018)
[14] https://github.com/netoptimizer/prototype-kernel/blob/master/kernel/Documentation/bpf/index.rst (May 2017)
[15] https://web.archive.org/web/20180520162550/https://dpdk.org/ (n.d.)

3.1 Threat Model and Security Notion

The overarching threat is undetectable domain impersonation (ex-post) by an attacker that is capable of compromising at least one CA and a sufficient number of CT logs to convince a TLS client into accepting a forged certificate with its associated SCTs and STHs. We assume that any illegitimately issued certificate would be detected by the legitimate domain owner or a proxy that monitors the logs. This means that an attacker must either provide a split view towards the victim or the owner/proxy. We also assume that TLS clients will query CT logs for certificate inclusion based on STHs that it acquires from the logs via plaintext mechanisms that packet processors can observe, and that there will be some other entities than challengers who process STHs via anonymity networks. We do not consider that a CA compromise alone may be detected, instead focusing solely on the split-view problem for CT logs.

3.1.1 Limitations. Our gossip mechanism is limited to STHs that packet processors can observe. As such, a client isolated by an attacker is not protected. We limit ourselves to attackers that act over a network some distance (in the sense of network path length) from a client in plaintext so that aggregation can take place. Our limitations and assumptions are further discussed in Section 6.

3.1.2 Attackers. Exceptionally powerful attackers can isolate clients, *but clients are not necessarily easy to isolate* for a significant number of relevant attackers. Isolation may require physical control over a device[16], clients may be using an anonymity network like Tor where path selection is inherently unpredictable [17], or an attacker simply cannot control sufficiently large parts of the network infrastructure to ensure that no aggregation takes place. This may in particular be the case if we consider a nation state actor attacking another nation state actor, the prevalence of edge security middleboxes, and that home routers or even closer packet processors like NICs could support aggregation. Any attacker that cannot account for these considerations are within our threat model.

3.1.3 Security Notion. An adaptive attacker may attempt to actively probe networks to discover aggregating packet processors with the goal of circumventing them to launch an undetected split view. This leads us to the key security notion for our mechanism: *aggregation indistinguishability*. An attacker should not be able to determine if a packet processor is aggregating STHs or not. The importance of aggregation indistinguishability motivates the design of our gossip mechanism into two distinct components: aggregation that takes place inline at packet processors and periodic off-path verification (log challenging) to verify whether STHs are consistent.

3.2 Packet Processor Aggregation

The packet processor component runs in-line, aggregating STHs by filtering and capturing relevant traffic for an off-path challenger. To setup the packet processor parameters are assigned based on implementation and STH source to specify how packets should be filtered to capture traffic containing STHs. A security parameter specifies the *probability* that the relevant traffic will be aggregated.

For each packet processor we need to take IP fragmentation and load into consideration. Without accounting for (intentional) IP fragmentation, an attacker can trivially fragment an STH to circumvent aggregation. The impact of multi-path fragmentation is discussed in Section 7.1. Depending on implementation, large traffic load may cause filtering and aggregation performance degradation resulting in a clear aggregation distinguisher that distant attackers can probe for. This is addressed by probabilistic filtering, potentially adjusting the probability based on load or simply aggregating with an acceptable probability for worst-case load. We emphasize that the security implications that relate to handling of IP fragmentation and aggregation indistinguishability are dependent on the intended packet processor as well as the aggregated STH source. Section 4 implements our design for two hardware targets and a given source.

3.3 Off-Path Log Challenging

A packet processor must be configured so that gossip-related traffic is aggregated to a challenger component that is *not run in-line*. Other than fetching STHs on its own off-path, the challenger reassembles IP fragments to track every STH that was probabilistically observed by its aggregator(s). The resulting set of STHs is used to challenge the logs to prove consistency periodically. It is paramount that these challenges cannot be linked to the either of the aggregating packet processor or the challenging challenger. This is to preserve aggregation indistinguishability, but also to achieve implicit gossip amongst different challenging components: it is hard to maintain a targeted split-view towards an unknown location, which is the case if the challenger fetches STHs and resolves proofs via an off-path [26]. This approach is similar to that of DoubleCheck by Alicherry and Keromytis [1], which uses Tor to detect man-in-the-middle attacks.

We say that a challenger is *off-path* if a log cannot link it to its aggregating packet processor or any challenger (including itself). For example, an anonymity network like Tor [17] could be used to this end, combined with delayed challenges to prevent timing correlations (hence not running in-line). The threat of probing with unique STHs to find aggregating paths is discussed in Section 7.1.

3.4 Multiple Aggregator-Challenger Instances

No actor controls all packet processors. As such, there will be many independent challengers that are reported to by its own packet processor(s). Occasionally aggregator-challenger instances may be implemented in the same box, e.g., if end-systems or security middleboxes are approached. In other cases an actor may have aggregators that report back to a logically centralized challenger, e.g., ASes, IXPs and ISPs that operate SDN-like infrastructures [22].

4 IMPLEMENTATIONS

While there are many types of packet processor that could support aggregation in different ways (further discussed in Section 7.2), our implementations specifically explore two approaches towards data plane programmability that together support a wide range of targets: P4 and XDP. First the plaintext source that our proof-of-concept implementations aggregate is introduced[17]. Next implementation details are outlined, and target-independent aggregation considerations elaborated upon for the selected plaintext source.

[16] For example as in the FBI-Apple San Bernardino case: https://web.archive.org/web/20180520135200/https://www.eff.org/cases/apple-challenges-fbi-all-writs-act-order (February 2016)

[17] All relevant code is available on GitHub: https://github.com/rgdd/ctga (June 2018)

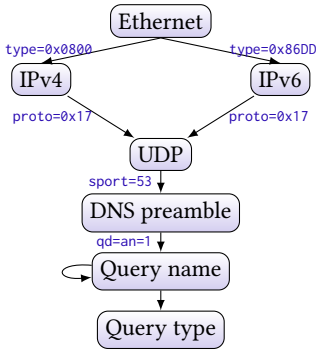


Figure 2: Parser graph

Finally, performance results are presented for two hardware targets. Notice that we do not implement the challenger: verification is relatively easy given an off-path and the aggregated STHs.

4.1 Plaintext Source

The fundamental assumption that aggregation-based gossip relies on is a plaintext source that packet processors can observe. The most applicable mechanism today is CT-over-DNS, which is hosted by Google for all Chrome-included logs. According to the draft by Laurie [33], a DNS STH response is an IN TXT resource record where the query domain is `sth.<log>.ct.googleapis.com`. We further restrict the format such that a response must be transported by UDP and contain (i) a single query, (ii) a single response, and (iii) no more than a threshold of bytes. These requirements facilitate data plane packet processing, and *must* be enforced by client software if gossip-based aggregation is adopted. Besides CT-over-DNS there are two other plaintext sources available: TLS ≤1.2 and OCSP. Dahlberg [43] discussed this in detail: TLS is inapplicable long-term because TLS 1.3 encrypts SCTs/STHs, and OCSP requires ASCII parsing which is hard at high rates. We would also add that it is challenging to use TCP because it is a stream-oriented protocol.

4.2 P4

Our proof-of-concept implementation targets the $P4_{16}$ `v1model` architecture and the behavioral model (Section 2.2.1), instantiating STH aggregation based on two fundamental building blocks: variable length headers and a hash function (e.g., based on CRC32).

4.2.1 Packet parsing. Figure 2 gives an overview of the headers that must be declared and parsed to aggregate DNS STHs. It is straight-forward to extract headers down to DNS, after which the parsing must continue in multiple stages: extract a fixed-width preamble that contains the number of questions (`qd`) and answers (`an`), loop to extract the query domain name, and finally extract the fixed remainder of the query (class and type). It is not possible to parse an arbitrary number of questions and answers in P4 because loops must be bound by a constant. This motivates the restricted CT-over-DNS format in Section 4.1. A variable-length domain name can only be parsed because it is a finite number of bytes and labels.

4.2.2 Packet processing. The parser outputs a parsed packet representation of valid headers that are processed by a number of match+action tables. Our ingress pipeline consists of a routing table that is always applied, and a log table that is only applied for DNS IN TXT packets. The table of known logs exact-match on the hash of a packet's query domain name, and upon hit it is marked for control-plane copying that the target's queue management system handles. While there are no special requirements on the hash function, bad spread increases false positives (i.e., copying overhead) under normal behaviour because TXT domain names may collide. IP fragments that are less than a threshold are also copy-marked. As an example of probabilistic filtering, our proof-of-concept supports every n^{th} packet copying; n is a register-stored security parameter.

4.3 XDP

The notion of STH aggregation is also expressible using eBPF. An eBPF program can be loaded into the Linux kernel's fast forwarding path, including before and after socket buffer allocation. Our proof-of-concept implementation targets the before use-case with XDP.

4.3.1 Packet parsing. Given that the same packet type should be parsed regardless of the implementation, Figure 2 is also applicable for XDP. Although it is possible to declare and parse custom headers, much of what we need is already available in kernel headers (e.g., IPv4 and UDP). The trickiest part of the parsing procedure is variable length fields, and the limitations are similar to P4: a loop must be unrolled at compile time, and thus be bound by a constant.

4.3.2 Packet processing. Because functionality is implemented as an imperative C-like program, XDP alternates between packet parsing and header processing. A packet that does not fit the parser graph can thus be routed towards its destination immediately. On a host this would be the normal networking stack, and on a router an outgoing interface. If a packet is determined to be a DNS IN TXT resource record with a single question-answer section, the parsed domain name is looked up in a hash map. Upon match, the packet is control-plane copied by inserting it into a ring buffer that a user space application can poll[18]. IP fragments that are less than a threshold are also copied, and n^{th} packet filtering is supported.

4.4 Other Considerations

Regardless of the approach towards implementing STH aggregation, it is vital to consider indistinguishability and handling of IP packets. We elaborate on each challenge and its implications below, relating the discussion to CT-over-DNS as the plaintext source.

4.4.1 Indistinguishability. The overarching packet processing is designed not to introduce any trivial distinguishers, such as dropping tiny fragments proactively. An implementation caveat, though, is that parser exceptions may be approached differently by programmable targets and developers. To provide aggregation indistinguishability, a packet that is malformed must neither be dropped nor altered (necessary but not sufficient). Given that a typical program often operates on lower-layer headers only, this is of particular importance when parsing UDP and DNS headers.

4.4.2 IP fragments and options. For data minimization, an IP fragment is only aggregated if it is less than a threshold. Therefore, a log client must reject STH packets that are too large. At the time

[18] https://github.com/cilium/cilium/blob/master/Documentation/bpf.rst (April 2018)

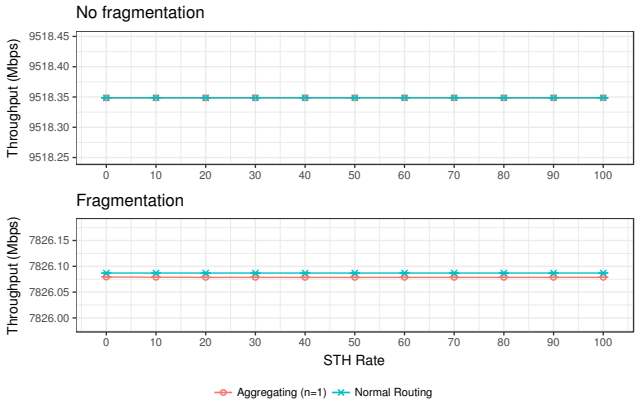
of writing a typical DNS STH is encoded as ≈170 bytes[19], and a 400 byte threshold would presumably be large enough to account for IP options, large domain names, and future STH extensions. However, the exact threshold must be specified unambiguously before deployment for clients, and should be treated as a security parameter by IPv6 packet processors due to variable length options. Notice that the privacy impact of aggregating small fragments have (presumably) little or no impact on legitimate traffic[20] [47]. After all, the de-facto minimum MTU has been *at least* 576 bytes for decades [9, 15]. This means that fragmentation is an anomaly rather than expected behaviour, and around 24 unique STHs per day and log should be found given sound STH frequencies for privacy [41].

4.5 Performance

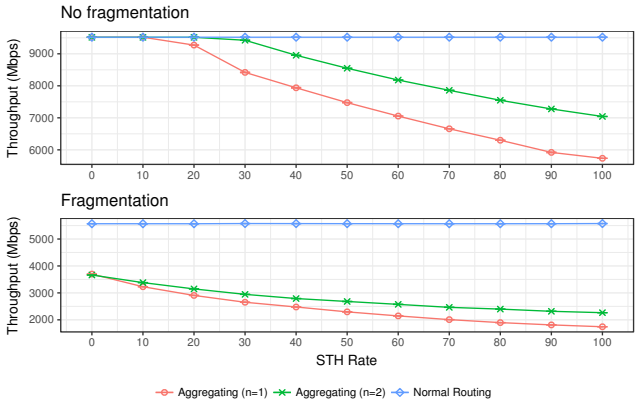
To evaluate performance and aggregation (in)distinguishability of our proof-of-concept implementations, a test-bed consisting of a traffic generator, a traffic receiver, and an aggregating target in between was set up. We used the open source network tester OSNT [2] for traffic generation and reception, replaying several different packet captures on a 10 Gb Xilinx NetFPGA SUME board[21]. The first target is also a 10 Gb Xilinx NetFPGA SUME board, but it runs an adapted version of our P4 reference implementation[22]. The second target is a net-next kernel v4.17.0-rc6[23] Linux machine that runs XDP on one core with a 10 Gb SFP+ X520 82599ES Intel card, a 3.6 GHz Intel Core i7-4790 CPU, and 16 GB of RAM at 1600 MHz (Hynix/Hyundai). We poll the ring buffer from a different core.

4.5.1 Experiments. Our experiments on a given target proceed as follows. First, a minimal program that routes the traffic from the generator to the receiver is loaded. This serves as a reference point, indicating how much throughput the target or the traffic generator and receiver can handle. Second, the minimal program is extended so that it aggregates small fragments and DNS STHs from 16 fictional logs whose domain names are assumed to be five labels of 1+16 bytes each. We chose excessively large domain names because it takes more time to parse and match, and normal sized DNS STHs. This resulted in 411 byte packets, although fragments are only 68 bytes. Our evaluated metric is throughput as the control plane copying match rate increases from 0–100% in intervals of 10, and we examine fragments and STHs separately every second for two minutes per interval. The background traffic of the respective experiments is non-fragments of the same size: 68 byte UDP packets for fragmentation, and otherwise 411 bytes DNS STHs that are unrecognized (miss in the table of known logs). In other words, the background traffic consumes as much resources as possible, but without being control-plane copied or changing the packet size.

4.5.2 Results. Figure 3a shows throughput as a function of match rate for the P4-enabled NetFPGA. While we were unable to observe any performance distinguisher between routing and aggregation of whole STHs, there is a minor difference for small fragments (7.5 Kbps). If a packet processor is physically isolated as in our benchmark, this is a non-negligible *program* distinguisher. This is not an issue for two reasons. First, within our threat model a packet processor cannot be isolated because the intended attacker is distant. Given the influence from noise traffic that competes with bandwidth, queueing, and other resources of the traversed packet processors, it would likely be non-trivial to pick up on in practise. Second, a program distinguisher is not an issue unless it uniquely identifies STH aggregation: *anything* could be running. This would not be the case if a performance metric changed *as a result of adjusting the STH rate* while aggregating.



(a) P4 NetFPGA



(b) XDP on a single core

Figure 3: Throughput as a function of match rate. In our threat model, aggregation indistinguishability is provided by P4-NetFPGA. For XDP it depends on how it is deployed.

Figure 3b shows throughput as a function of match rate for the single core XDP target. As long as the STH rate is less than 10%, we were unable to observe any trivial aggregation distinguisher for whole STHs. If the security parameter n is set such that every other packet is copied instead, then aggregation indistinguishability is also provided for 20% STH rate. This suggests that control plane

[19] We queried all Chrome-included logs via DNS, inspecting the returned responses.
[20] https://web.archive.org/web/20180612113649/https://tools.cisco.com/security/center/viewIpsSignature.x?signatureId=1206& (January 2006)
[21] https://web.archive.org/web/20170702013300/http://store.digilentinc.com/netfpga-sume-virtex-7-fpga-development-board/ (n.d.)
[22] Functionally is the same, expect that the P4 SUME architecture have yet to support variable length headers. To overcome this deficiency, we assumed fixed-length domain names whose labels are extracted one at a time in five distinct parser states.
[23] https://git.kernel.org/pub/scm/linux/kernel/git/davem/net-next.git/ (May 2018)

copying is the most expensive operation for whole STHs, and by fine-tuning n a probabilistic filtering mechanism can combat evident performance distinguishers. However, adjusting n is insufficient for small fragments: on our system one-core line rate of small packets is near 6 Gbps, but despite nothing being copied throughput drops down to 3.7 Gbps. This program distinguisher might be somewhat unique, but it is not necessarily a problem. For example, 10 Gbps line-rate could be achieved by using three cores (as opposed to one). Moreover, line rate on an end-system is typically less than 1 Gbps.

Figure 3 thus shows that P4 and XDP can provide aggregation indistinguishability within our threat model (either with or without extra configuration depending on the target). Our proposal supports stress-testing with up to 10 Gbps of STH-only traffic, although we were unable to test aggregation of small fragments at rates larger than 7.8 Gbps. This is a deficiency that stems from traffic generation and reception. It should be noted that these performance numbers are mainly interesting for *aggregation indistinguishability*: it would be abnormal network behaviour to see, e.g., 10% of the traffic being DNS STHs or fragments, and for the expected case of non-fragmented STHs and lower rates aggregation is basically for free. Preliminarily, we also explored if *latency* is a distinguisher for P4-NetFPGA (see Appendix A). It appears that STH intervals are indistinguishable, but the presence of a program is not ($\Delta \leq 5$ μs).

5 ESTIMATED IMPACT OF DEPLOYMENT

We conducted traceroute measurements on the RIPE Atlas platform to evaluate the effectiveness of aggregation-based gossip. Ultimately this is used to quantify the direct protection against split-views for 40% of the IPv4 space (and hence a large portion of all TLS clients on the Internet) by looking at coverage as $1 \ldots n$ actors run packet processors that aggregate to their own off-path challenger instances. An IPv4 address is considered covered if its on a path that involves at least one aggregator when fetching an STH, and our traceroute data set is used to determine these paths towards real (Google) and fictitious (NORDUnet) CT-over-DNS resolvers. Aggregating actors, namely ASes and IXPs, are selected based on two different top-ranked criteria. Note that Chuat *et al.* [12] also used network measurements to evaluate properties of their gossip mechanism, rather than simulating a network scenario where the log goes rogue to determine if and when this is detected (which depends more on assumptions related to the model than the proposed protocol).

5.1 RIPE Atlas Data Set

Our traceroute measurements can be downloaded by anyone on the RIPE Atlas platform. Use the following measurement identifiers: 11603880–11603884, 11784033–11784042, and 11826645–11826649.

5.1.1 Target selection. We targeted Google's authoritative CT-over-DNS server. This is self-explanatory, given that it is the most realistic plaintext mechanism today. As a secondary target we also included SUNET's Plausible CT log. Unlike Google who operates a world-wide infrastructure, SUNET is part of NORDUnet which is a Nordic network provider that interconnects education networks. We hypothesized that there might be interesting differences in the observed path characteristics, possibly affecting client protection.

5.1.2 Probe selection. The goal of our probe selection process was to maximize the number of unique ASes (which will represent blocks of IP addresses that we can evaluate coverage for). The scope of our search was reduced to IPv4 because many probes support it, and for redundancy the two most stable probes in each unique AS were selected. We based the stable criteria on the RIPE Atlas tag `system-ipv4-stable-n`, such that a probe got the highest priority if n=90 days. While many ASes had too few probes to support redundancy, we ended up requesting 4604 probes. After removing the redundant probes that delivered the fewest amount of traceroute results, there were little or no failures amongst the remaining 3512 (Google) and 3488 (NORDUnet) probes: around 100 probes failed at least once, and among those 24 as well as 17 probes (respectively) failed more than once. This means that the reliability of RIPE Atlas platform is remarkably high, and thus it is unnecessary to account for failures while analyzing the results in Sections 5.3–5.4.

5.1.3 Duration and measurement settings. For all probes we scheduled a daily traceroute towards Google and NORDUnet. Our measurements towards Google started on March 10 2018, and ended on March 30 2018. Ten days later on March 20, we started another measurement towards NORDUnet that ended on April 9 2018. We used the RIPE Atlas default traceroute settings, resulting in ICMP port 80 with default spread and Paris traceroute enabled[24] (value 16). The response timeout was set to 4000 ms for three 48 byte packets and 32 max hops. We also hard-coded the targeted IP addresses because not all probes support DNS lookups. To verify that the mapping from domain name to IP address remained the same for Google's authoritative CT-over-DNS server, we conducted a daily santiy-check[25] from 128 worldwide probes that resolved `ctns.googleapis.com` ≡ `216.239.34.64` on the probes. An employee at SUNET verified that `plausible-fe1.ct.nordu.net` ≡ `194.68.13.48` would remain stable throughout our experiments.

5.2 Other Data Sets

The traceroute data set in Section 5.1 contains lists of IP addresses. Since we are interested in the actors that control the corresponding packet processors, i.e., which actors are on a given path, we mapped each IP address to an AS number and/or IXP identifier using public data sets from Routeviews[26] and CAIDA[27]. We also relied on RIPE Atlas probe metadata to map probes to AS numbers[28], CAIDA's largest AS rank to select aggregators while computing coverage[29], and Routeviews' data set to annotate each probe with the IPv4 space of its AS (Section 5.3.3). Due to a number of reasons such annotations are imperfect. For example, there are overlapping IP blocks in the Routeviews data set, and an IP address may be unused or reused. Nevertheless, it gives a decent idea of how significant it is for an aggregator to cover a given probe.

[24] https://web.archive.org/web/20180511201452/https://paris-traceroute.net/ (n.d.)
[25] RIPE Atlas measurement identifiers: 11603871 and 11793938.
[26] The Routeviews MRT format RIBs and UPDATEs Dataset, 2018-03-12 14:00, http://archive.routeviews.org/bgpdata/2018.03/RIBS/
[27] The CAIDA UCSD IXPs Dataset, February 2018, https://www.caida.org/data/ixps/
[28] https://atlas.ripe.net/docs/api/v2/reference/#!/probes/probe$_{l}$ist$_{g}$et (n.d.), accessed REST API on 2018-04-06
[29] http://as-rank.caida.org/api/v1 (n.d.), accessed REST API on 2018-04-06

5.3 Results

Let an AS path be the set of traversed ASes from a probe *before* reaching Google or NORDUnet, and an IXP path the set of traversed IXPs. Now we use our RIPE Atlas data set to examine path length, path stability, and client coverage as n actors aggregate STHs.

5.3.1 Path length. Figure 4 shows that an AS path tends to be one hop longer towards NORDUnet. This is evident because there is a rough off-by-one offset on the x-axis: 27.0%, 51.8% and 15.7% of all paths traverse one, two and three ASes towards Google, while 28.7%, 45.3% and 15.6% of all paths traverse two, three and four ASes towards NORDUnet. A similar trend of greater path lengths towards NORDUnet can be observed for IXPs. For example, 74.0% of all paths traverse no IXP towards Google, but 58.5% of all paths traverse a single IXP towards NORDUnet. We explain these results by referring to the infrastructural differences of our targets: Google is a worldwide actor, which means that an average path should be shorter than compared to a region-restricted actor like NORDUnet.

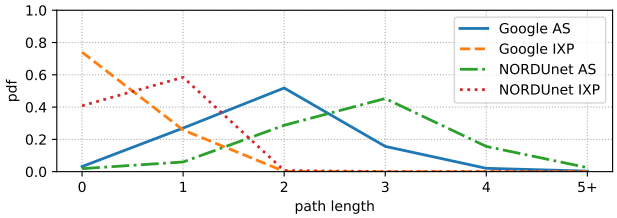


Figure 4: Path length towards Google and NORDUnet.

5.3.2 Path stability. Figure 5 shows path stability for ASes and IXPs. The x-axis represents a number of distinct paths towards a target, and the y-axis a fraction of probes that this applied for. Both AS and IXP paths tend to change infrequently: while 14.8% (Google) and 26.9% (NORDUnet) of all probes had at least two distinct AS paths, the stability is even greater for traversed IXPs. A path is likely less stable towards NORDUnet because more actors are involved in the process of getting there. Nevertheless, the observed paths remained stable for both targets throughout our measurements.

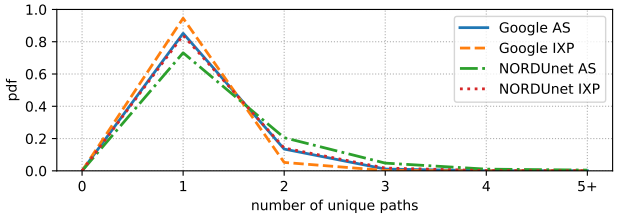


Figure 5: Path stability towards Google and NORDUnet.

5.3.3 Coverage. To look at the *concrete* protection of our gossip mechanism as $1 \ldots n$ actors opt-in, we compiled different lists of ASes and IXPs that are assumed to host aggregator-challenger instances. First these lists are used to determine the fraction of RIPE Atlas probes that traversed at least one aggregator, thereby receiving split view protection because every off-path challenger must be on the same view (otherwise an inconsistency is detected by someone). Second, we annotated each probe with the IPv4 space of its AS. This gives us an understanding of how well 40% of the IPv4 space is covered (biased towards Europe and the US due to using RIPE Atlas). The lists of aggregating actors are based on:

Pop A popularity rank derived form our own measurements. For example, if actor x, y and z are traversed by three, four and five probes, then $n = 2$ suggests that actors y and z should aggregate to cover as many probes as possible. This is not necessarily optimal, e.g., z might already cover the probes of y but not x. Nevertheless, it is a simple approach that works for the selection of aggregating ASes *and* IXPs.

CAIDA An AS rank derived from several Internet topology data sets. ASes receive high rank if, according to CAIDA, they are globally influential. This is based on factors such as size, customer cone, and inferred AS business relationships[30].

Figure 6 shows probe coverage as $1 \ldots 1024$ actors opt-in for aggregation-based gossip. An evident pattern is that the probes are better protected against split views provided by NORDUnet than Google. This result is related to path length: given that more ASes and IXPs tend to be traversed, the likelihood of shared vantage points and aggregation increases. While it is good to cover RIPE Atlas probes, the ultimate goal is to understand the direct protection of *clients*. As explained already, this lead us to add weights to each probe. The results are shown in Figure 7, and coverage derived from Pop tends to decrease: AS CAIDA outperforms AS Pop for Google despite our attempt to specifically cover as many probes as possible, while AS Pop still performs better than AS CAIDA for NORDUnet. If CAIDA's top-32 aggregated, then client protection would be significant for Google (31.6%) and NORDUnet (58.1%) both. IXP aggregation would also be significant, but it appears to better cover small ASes (cf. weighted and unweighted IXP Pop).

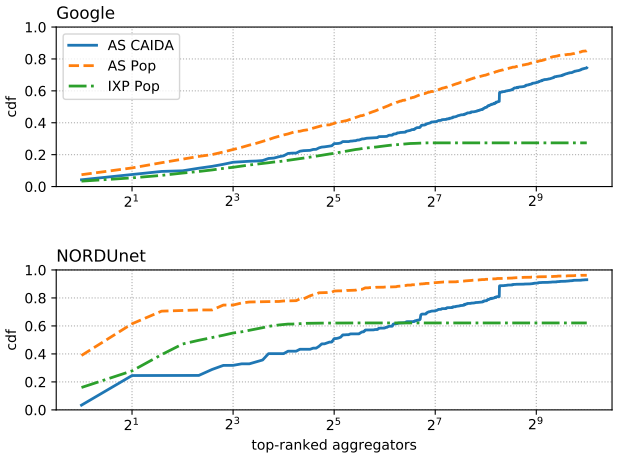


Figure 6: Probe coverage as a function of aggregation opt-in.

[30] https://web.archive.org/web/20180608161146/http://as-rank.caida.org/about (n.d.)

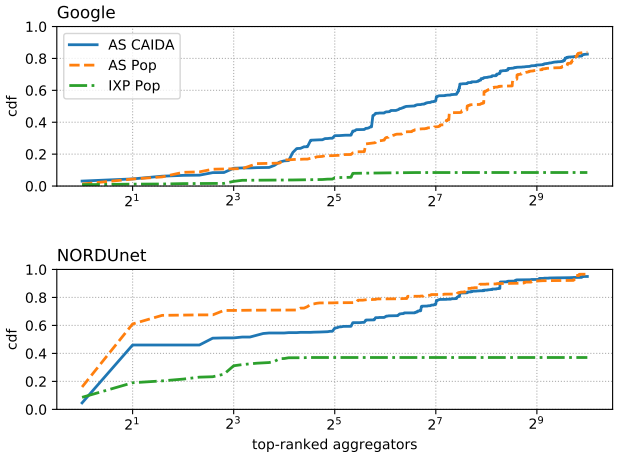


Figure 7: Client coverage as a function of aggregation opt-in.

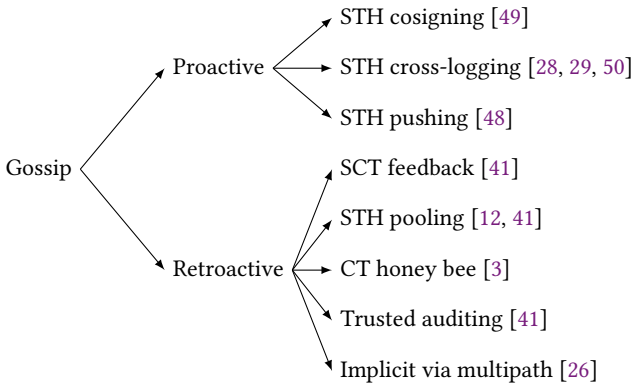


Figure 8: A categorization of approaches towards CT gossip.

5.4 Lessons learned

Despite Google's wide reach, a vast majority of all clients traverse *at least* one AS which could aggregate. It is relatively rare to traverse IXPs towards Google but not NORDUnet. The fact that paths are stable indicate that an (un)protected client remains (un)protected. Therefore, the time until split view detection could be long *if* it is possible to find an unprotected client, increasing the importance of aggregation indistinguishability. We finally identified vantage points that are commonly traversed using Pop, and these vantage points are represented well by CAIDA's independent AS ranking. Little opt-in from ASes and IXPs provides significant protection against an attacker that is somewhat close to a client (cf. world-wide infrastructure of Google), and although we got better coverage for NORDUnet any weak attacker would likely approach the coverage properties of Google by renting infrastructure that is nearby. Similarly, any modestly sophisticated attacker would circumvent any IXP aggregator by detecting the IXP itself. Aggregating IXPs may still be useful though to detect split views due to other reasons than malicious attackers, as discussed in Section 7.4.

6 RELATED WORK

Figure 8 categorizes earlier approaches that resemble CT gossip based on if gossiping is *proactive* or *retroactive*. An approach is proactive if gossip takes place *before* SCTs and/or STHs reach the broader audience of clients. Syta *et al.* proposed proactive witness cosigning, in which an STH is collectively signed by a *large* number of witnesses and at most a fraction of those can be faulty to ensure that a benevolent witness observed an STH [49]. STH cross-logging is similar in that an STH must be proactively disclosed in another transparency log to be trusted, avoiding any additional cosigning infrastructure at the cost of reducing the size and diversity of the witnessing group[31]. Tomescu and Devadas [50] suggested a similar cross-logging scheme, but split-view detection is instead reduced to the difficulty of forking the Bitcoin blockchain (big-O cost of downloading all block headers as a TLS client). Finally, STH pushing assumes a secure channel to ensure that a group of TLS clients get the same STH history pushed from a shared trusted third-party [48].

A gossip mechanism is retroactive if gossip takes place *after* SCTs and/or STHs reach the broader audience of clients. Chuat *et al.* proposed that TLS clients and TLS servers be modified to pool exchanged STHs and associated consistency proofs [12]. Nordberg *et al.* continued this line of work, suggesting privacy-preserving client-server pollination of fresh STHs [41]. It was also proposed that clients feedback SCTs and certificate chains on server revisits, and that trusted auditor relationships could be engaged if privacy does not matter. In a sense the latter is similar to the formalized client-monitor gossip of Chase and Meiklejohn [11], as well as the CT honey bee project where a client-enabled process fetches and submits STHs to a pre-compiled list of auditors [3]. Laurie suggested that a client can resolve privacy-sensitive SCTs to privacy-insensitive STHs via DNS [33] (which are easier to gossip). Private information retrievals could likely achieve something similar [36]. Assuming that TLS clients are indistinguishable while interacting with the log, split-view detection can also be implicit as proposed by Gunn *et al.* for the verifiable key-value store in CONIKS [26, 39].

Given that aggregation-based gossip takes place after an STH is issued, it is a retroactive approach. As such, we cannot protect an isolated client from split-views [49]. Similar to STH pooling and STH pollination, we rely on client-driven communication and an existing infrastructure of packet processors to aggregate (cf. using TLS servers as pools). Our off-path verification is based on the same multi-path probing and indistinguishability assumptions as Gunn *et al.* [1, 26, 51]. Further, given that aggregation is application neutral and deployable on hosts, it could provide gossip *for* the CT honey bee project if it used a plaintext mechanism. Our approach coexists well with witness cosigning and cross-logging since the threat models are different, but not necessarily with STH pushing if the secure channel is encrypted because clients are unlikely to fetch anything that is already provided by a trusted third-party.

[31] https://mailarchive.ietf.org/arch/msg/trans/7eL7nu3eaAxEf4snLj7w0TUnrwE (January 2017), accessed 2018-06-13. See also Google's 'minimal gossip' in footnote 4.

7 DISCUSSION

The benefit of our gossip mechanisms compared to browsing-centric and vendor-specific approaches is that it is application neutral. For example, on the web our approach covers a plethora of HTTPS clients, ranging from niche web browsers to command line tools and embedded libraries that are vital to protect but yet lack the resources of major browser vendors [4, 16]. Being application neutral means that our approach works for any type of transparency log, including binary transparency and generalizations of CT [21, 29].

7.1 Assumptions and Limitations

Aggregation-based gossip is limited to network traffic that packet processors can observe. The strongest type of attacker in this setting—who can completely isolate a client—trivially defeats our gossip mechanism and other retroactive approaches (see Section 6). A weaker attacker cannot isolate a client, but is located nearby in a network path length sense. This limits the opportunity for packet processor aggregation, but an attacker cannot rule it out given aggregation indistinguishability. Section 4 showed based on performance that it is non-trivial to distinguish between (non-)aggregating packet processors on two different targets using P4 and XDP. Off-path challengers must also be indistinguishable from one another to achieve *implicit gossip*. While we suggested the use of anonymity networks like Tor, a prerequisite is that this is in and of itself not an aggregation distinguisher[32]. Therefore, we assume that other entities also use off-paths to fetch and verify STHs. The fact that a unique STH *is not audited* from an off-path could also be an aggregation distinguisher. To avoid this we could rely on a verifiable STH history[33] and wait until the next MMD to audit or simply monitor the full log so that consistency proofs are unnecessary.

The existence of multiple network paths are fundamental to the structure and functioning of the Internet. A weak attacker may use IP fragmentation such that each individual STH fragment is injected from a different location to make aggregation harder, approaching the capabilities of a stronger attacker that is located closer to the client. This is further exacerbated by the deployment of multi-path transport protocols such as MPTCP, which can also be fragmented. Looking back at our RIPE Atlas network measurements in Section 5, the results towards Google's world-wide infrastructure better represent an active attacker that takes *some* measures to circumvent aggregation by approaching a client nearby the edge. Given that the likelihood of aggregation is high if *any* IXP is present (Figures 6–7), we suspect that deployment of aggregation at popular and well-connected IXPs are the most likely to be circumvented.

7.2 Deployment

Besides aggregating at strategic locations in the Internet's backbone, ISPs and enterprise networks have the opportunity to protect all of their clients with relatively little effort. Deployment of special-purpose middleboxes are already prevalent in these environments, and then the inconvenience of fragmentation tends to go away due to features such as packet reassembly. Further, an attacker cannot trivially circumvent the edge of a network topology—especially not if aggregation takes place on an end-system: all fragments are needed to reassemble a packet, which means that multi-path fragmentation is no longer a threat. If aggregation-based gossip is deployed on an end-system, STHs *could* be hooked using other approaches than P4/XDP. For example, shim-layers that intercepts TLS certificates higher up in the networking stack were already proposed by Bates *et al.* [5] and O'Neill *et al.* [42]. In this setting an end-system is viewed as the aggregating packet processor, and it reports back to an off-path challenger that may be a local process running on the same system or a remote entity, e.g., a TelCo could host challengers that collect aggregated STHs from smartphones.

While we looked at programming the data plane of physical packet processors to instantiate the aggregation step, there are other options and locations for STH aggregation to take place:

- Hypervisors and software switches [46] that reside inside virtualized environments could protect many virtual hosts.
- DNS servers are ideal for aggregating STHs requested via CT-over-DNS, e.g., dump the cache periodically to a challenger.
- Similar to DNS servers, so called Tor exits operate DNS caches for all clients that perform DNS queries over Tor.[34]
- NAT gateways—especially carrier-grade NAT—are naturally isolating clients behind network choke-points.

In other words, P4 and XDP are *instantiation examples* of the aggregation step. Custom hardware description languages, simply C for some special-purpose middleboxes, or OS-level shim-layers may be more appropriate depending on the used plaintext source, the target, and the surrounding network topology.

7.3 Retroactive Gossip Benefits From Plaintext

The proliferation of middleboxes that inspect and shape packet headers contradict the intended design of the Internet. As opposed to a dumb core that forwards IP packets, network and security functions are often embedded which cause complex processing dependencies and protocol ossification [30]. Since middleboxes that inspect and modify packets have caused security and protocol issues [20, 32], the current mindset is to encrypt everything [32]. Our work is controversial because it goes against this mindset and advocates that STHs should be communicated in plaintext. We argue that this makes sense in the context of STHs due to the absence of privacy concerns and because the entire point of gossip is to make STHs *available* (rather than being end-to-end only). It is also beneficial if STHs can be easily parsed in-line to help packet processors extract them from packets efficiently. The idea of intentionally exposing information to the network is not new; MPQUIC is designed like this to support flexible traffic shaping [13].

[32] For reference, Tor has about two million daily users: https://web.archive.org/web/20180522081849/https://metrics.torproject.org/userstats-relay-country.html (February–May 2018). Note that low-latency anonymity networks like Tor are susceptible to traffic confirmation and correlation attacks where the attacker observes traffic from the packet processor and is in control of the response from the CT logs. A strictly isolated packet processor may not be able to hide that it is challenging the logs.

[33] https://web.archive.org/web/20170806160119/https://mailarchive.ietf.org/arch/msg/trans/JbFiwO90PjcYzXrEgh-Y7bFG5Fw (May 2017).

[34] In addition to implicit gossip, notice that an STH requested by a challenger in plaintext via Tor may be aggregated again by a different aggregator-challenger instance.

While we looked at CT-over-DNS as a plaintext source, there is a push towards DNS-over-TLS[35] and DNS-over-HTTPS[36]. Wide use of TLS may undermine our approach towards gossip, but ironically the security of TLS could be jeopardized unless gossip is deployed. In other words, long term gossip is an essential component of CT and other transparency logs to avoid becoming yet another class of trusted third-parties. If proactive approaches such as witness cosigning are rejected in favour of retroactive mechanisms, then ensuring that STHs are widely spread and easily accessible is vital. With care taken to ensure that a recent STH is not privacy sensitive (refer to the discussion of Section 2.1.2), it need not be confidential. Secure channels also provide integrity and replay protection, but an STH is already signed by logs and freshness is covered by MMDs as well as issue frequency to protect privacy. A valid argument against exposing any plaintext to the network is protocol ossification. We emphasize that our design motivates why packet processors should fail open: otherwise there is no aggregation indistinguishability.

7.4 Indistinguishability and Herd Immunity

An attacker that gains control over a CT log is bound to be more risk averse than an attacker that compromises a CA. There is an order of magnitude less logs than CAs (few dozens as opposed to hundreds), and client vendors are likely going to be exceptionally picky when it comes to accepted and rejected logs. We have already seen examples of this, including Google Chrome disqualifying logs that made mistakes: Izenpe used the same key for production and testing[37], and Venafi suffered from an unfortunate power outage[38]. Risk averse attackers combined with packet processors that are aggregation indistinguishable may lead to *herd immunity*: despite a significant fraction of clients that lack aggregators, a sense of indirect protection can be provided because the risk of eventual detection is unacceptable to many attackers. Hof and Carle [29] and Nordberg *et al.* [41] also discussed herd immunity briefly before us.

Ironically, announcing gossip-based aggregation 'as-a-service' through marketing may be beneficial for herd immunity and clients despite violating the notion of aggregation indistinguishability. This is especially the case for aggregators on the edge that cannot be bypassed, such as ISPs and enterprise networks. However, actively probing for and verifying whether aggregation is in place is still a key capability for any attacker. For example, an announcement may not follow through or only a subset of all paths have an aggregator.

8 CONCLUSIONS

Soon wide spread modifications of TLS clients are inevitable to support CT gossip. We proposed that these modifications include challenging the logs to prove certificate inclusion based on STHs fetched in plaintext, enabling the traversed packet processors to assist in split view detection retroactively by aggregating STHs that are verified for consistency periodically using an off-path. Beyond being an application neutral approach that is complementary to proactive gossip, a compelling aspect is that core packet processors are used (rather than clients) as a key building block to realize implicit gossip; should a consistency issue arise, it is already in the hands of an entity that is well equipped to investigate the cause manually. Considering that far from all TLS clients are backed by big browser vendors—not to mention other use-cases of CT in general—it is likely a long-term win to avoid pushing complex gossip logic into all the different types of clients when compared to the order of magnitude fewer packet processors that can aggregate to off-path challengers. While taking the risk of ossification into account by suggesting that packet processors fail open to provide an essential security property—namely aggregation indistinguishability—our approach offers rapid incremental deployment with high impact on a significant fraction of Internet users. The notion of aggregation indistinguishability also motivates why our design does not boost any performance or privacy aspects by caching CT-related traffic.

[35] https://web.archive.org/web/20180422194047/https://security.googleblog.com/2018/04/dns-over-tls-support-in-android-p.html (April 2018)
[36] https://web.archive.org/web/20180512125541/https://blog.cloudflare.com/dns-resolver-1-1-1-1/ (April 2018)
[37] https://groups.google.com/a/chromium.org/forum/#!topic/ct-policy/qOorKuhL1vA (May 2016), accessed 2018-06-09
[38] https://groups.google.com/a/chromium.org/forum/#!topic/ct-policy/KMAcNT3asTQ (March 2017), accessed 2018-06-09

ACKNOWLEDGMENTS

We would like to thank Stefan Alfredsson and Philipp Winter for their RIPE Atlas credits, as well as Jonas Karlsson and Ricardo Santos for helping out with the NetFPGA setup. Rasmus Dahlberg, Tobias Pulls, and Andreas Kassler received funding from the HITS research profile funded by the Swedish Knowledge Foundation.

REFERENCES

[1] Mansoor Alicherry and Angelos D. Keromytis. 2009. DoubleCheck: multi-path verification against man-in-the-middle attacks. In *Proceedings of the 14th IEEE Symposium on Computers and Communications (ISCC)*, 557–563.
[2] Gianni Antichi et al. 2014. OSNT: Open source network tester. *IEEE Network*, 28, 5, 6–12.
[3] Andrew Ayer. 2018. Lightweight program that pollinates STHs between certificte transparency logs and auditors. (2018). https://github.com/SSLMate/ct-honeybee.
[4] Michael Backes, Sven Bugiel, and Erik Derr. 2016. Reliable third-party library detection in Android and its security applications. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 356–367.
[5] Adam M. Bates, Joe Pletcher, Tyler Nichols, Braden Hollembaek, Dave Tian, Kevin R. B. Butler, and Abdulrahman Alkhelaifi. 2014. Securing SSL certificate verification through dynamic linking. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 394–405.
[6] Alex Bavelas. 1950. Communication patterns in task-oriented groups. *The Journal of the Acoustical Society of America*, 22, 6, 725–730.
[7] Pat Bosshart et al. 2014. P4: Programming protocol-independent packet processors. *Computer Communication Review*, 44, 3, 87–95.
[8] Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard, Fernando Mujica, and Mark Horowitz. 2013. Forwarding metamorphosis: fast programmable match-action processing in hardware for SDN. In *Proceedings of the ACM SIGCOMM*, 99–110.
[9] R. Braden. 1989. Requirements for Internet hosts—communication layers. RFC 1122. IETF. 116 pp.
[10] Gordon Brebner. 2015. P4 for an FPGA target. In *1st P4 Workshop*. https://p4workshop2015.sched.com/event/3ZQA/p4-for-an-fpga-target.
[11] Melissa Chase and Sarah Meiklejohn. 2016. Transparency overlays and applications. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 168–179.
[12] Laurent Chuat, Pawel Szalachowski, Adrian Perrig, Ben Laurie, and Eran Messeri. 2015. Efficient gossip protocols for verifying the consistency of certificate logs. In *IEEE Conference on Communications and Network Security (CNS)*, 415–423.
[13] Quentin De Coninck and Olivier Bonaventure. 2017. Multipath QUIC: Design and evaluation. In *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, 160–166.
[14] Scott A. Crosby and Dan S. Wallach. 2009. Efficient data structures for tamper-evident logging. In *18th USENIX Security Symposium*, 317–334.
[15] S. Deering and R. Hinden. 2017. Internet protocol version 6 (IPv6) specification. RFC 8200. IETF. 42 pp.

[16] Erik Derr, Sven Bugiel, Sascha Fahl, Yasemin Acar, and Michael Backes. 2017. Keep me updated: An empirical study of third-party library updatability on Android. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2187–2200.

[17] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. 2004. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, 303–320.

[18] Benjamin Dowling, Felix Günther, Udyani Herath, and Douglas Stebila. 2016. Secure logging schemes and certificate transparency. In *21st European Symposium on Research in Computer Security (ESORICS)*, 140–158.

[19] Zakir Durumeric, James Kasten, Michael Bailey, and J. Alex Halderman. 2013. Analysis of the HTTPS certificate ecosystem. In *Proceedings of the 2013 Internet Measurement Conference*, 291–304.

[20] Zakir Durumeric, Zane Ma, Drew Springall, Richard Barnes, Nick Sullivan, Elie Bursztein, Michael Bailey, J. Alex Halderman, and Vern Paxson. 2017. The security impact of HTTPS interception. In *24th Annual Network and Distributed System Security Symposium (NDSS)*.

[21] Adam Eijdenberg, Ben Laurie, and Al Cutter. 2015. Verifiable data structures. Design document. Google Inc. 6 pp. https://github.com/google/trillian/blob/master/docs/VerifiableDataStructures.pdf.

[22] Nick Feamster, Jennifer Rexford, and Ellen W. Zegura. 2014. The road to SDN: An intellectual history of programmable networks. *Computer Communication Review*, 44, 2, 87–98.

[23] Adrienne Porter Felt, Richard Barnes, April King, Chris Palmer, Chris Bentzel, and Parisa Tabriz. 2017. Measuring HTTPS adoption on the web. In *26th USENIX Security Symposium*, 1323–1338.

[24] Oliver Gasser, Benjamin Hof, Max Helm, Maciej Korczynski, Ralph Holz, and Georg Carle. 2018. In log we trust: Revealing poor security practices with certificate transparency logs and Internet measurements. In *19th International Conference on Passive and Active Measurement (PAM)*, 173–185.

[25] Andy Gospodarek and Jesper Dangaard Brouer. 2017. XDP for the rest of us. In *Netdev 2.2: The Technical Conference on Linux Networking*. Workshop session. https://www.netdevconf.org/2.2/session.html?gospodarek-xdp-workshop.

[26] Lachlan J. Gunn, Andrew Allison, and Derek Abbott. 2017. Safety in numbers: Anonymization makes keyservers trustworthy. In *10th Workshop on Hot Topics in Privacy Enhancing Technologies*, 1–2.

[27] András Hajnal, Eric C Milner, and Endre Szemerédi. 1972. A cure for the telephone disease. *Canadian Mathematical Bulletin*, 15, 3, 447–450.

[28] Benjamin Hof. 2017. STH cross logging. Internet-draft draft-hof-trans-cross-00. Work in progress. IETF. 6 pp.

[29] Benjamin Hof and Georg Carle. 2017. Software distribution transparency and auditability. *CoRR*, abs/1711.07278, 1–14.

[30] Michio Honda, Yoshifumi Nishida, Costin Raiciu, Adam Greenhalgh, Mark Handley, and Hideyuki Tokuda. 2011. Is it still possible to extend TCP? In *Proceedings of the 11th ACM SIGCOMM Internet Measurement Conference (IMC)*, 181–194.

[31] D. Kumar et al. 2018. Tracking certificate misissuance in the wild. In *IEEE Symposium on Security and Privacy (SP)*, 288–301.

[32] Adam Langley et al. 2017. The QUIC transport protocol: Design and Internet-scale deployment. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, 183–196.

[33] Ben Laurie. 2016. Certificate transparency over DNS. Design document draft-ct-over-dns-01. Google Inc. https://github.com/google/certificate-transparency-rfcs/blob/master/dns/draft-ct-over-dns.md.

[34] Ben Laurie, Adam Langley, and Emilia Kasper. 2013. Certificate transparency. RFC 6962. IETF. 27 pp.

[35] Ben Laurie, Adam Langley, Emilia Kasper, Eran Messeri, and Rob Stradling. 2017. Certificate transparency version 2.0. Internet-draft draft-ietf-trans-rfc6962-bis-28. Work in progress. IETF. 55 pp.

[36] Wouter Lueks and Ian Goldberg. 2015. Sublinear scaling for multi-client private information retrieval. In *19th International Conference on Financial Cryptography and Data Security*, 168–186.

[37] Antonis Manousis, Roy Ragsdale, Ben Draffin, Adwiteeya Agrawal, and Vyas Sekar. 2016. Shedding light on the adoption of Let's Encrypt. *CoRR*, abs/1611.00469, 1–14.

[38] Steven McCanne and Van Jacobson. 1993. The BSD packet filter: A new architecture for user-level packet capture. In *Proceedings of the Usenix Winter Technical Conference*, 259–270.

[39] Marcela S. Melara, Aaron Blankstein, Joseph Bonneau, Edward W. Felten, and Michael J. Freedman. 2015. CONIKS: Bringing key transparency to end users. In *24th USENIX Security Symposium*, 383–398.

[40] Ralph C. Merkle. 1987. A digital signature based on a conventional encryption function. In *Advances in Cryptology (CRYPTO)*, 369–378.

[41] Linus Nordberg, Daniel Kahn Gillmor, and Tom Ritter. 2017. Gossiping in CT. Internet-draft draft-ietf-trans-gossip-05. Work in progress. IETF. 57 pp.

[42] Mark O'Neill et al. 2017. Trustbase: an architecture to repair and strengthen certificate-based authentication. In *26th USENIX Security Symposium*, 609–624.

[43] Rasmus Dahlberg. 2018. *Aggregating Certificate Transparency Gossip Using Programmable Packet Processors*. Master Thesis. Karlstad University, 1–69.

[44] Intel Corporation. [n. d.] Intel Ethernet Switch FM600 Series: 10/40 GbE Low Latency Switching Silicon. Product specification. 2 pp. https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/ethernet-switch-fm6000-series-brief.pdf.

[45] Netronome Systems Inc. [n. d.] Programming NFP with P4 and C. White paper. 9 pp. https://www.netronome.com/media/redactor_files/WP_Programming_with_P4_and_C.pdf.

[46] Muhammad Shahbaz, Sean Choi, Ben Pfaff, Changhoon Kim, Nick Feamster, Nick McKeown, and Jennifer Rexford. 2016. PISCES: A programmable, protocol-independent software switch. In *Proceedings of the ACM SIGCOMM 2016 Conference*, 525–538.

[47] Colleen Shannon, David Moore, and Kimberly C. Claffy. 2002. Beyond folklore: Observations on fragmented traffic. *IEEE/ACM Transactions on Networking*, 10, 6, 709–720.

[48] Ryan Sleevi and Eran Messeri. 2017. Certificate transparency in Chrome: Monitoring CT Logs consistency. Design document. Google Inc. 5 pp. https://docs.google.com/document/d/1FP5J5Sfsg0OR9P4YT0q1dM02iavhi8ix1mZlZe_z-ls/edit?pref=2&pli=1.

[49] Ewa Syta, Iulia Tamas, Dylan Visher, David Isaac Wolinsky, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ismail Khoffi, and Bryan Ford. 2016. Keeping authorities "honest or bust" with decentralized witness cosigning. In *IEEE Symposium on Security and Privacy (SP)*, 526–545.

[50] Alin Tomescu and Srinivas Devadas. 2017. Catena: efficient non-equivocation via Bitcoin. In *IEEE Symposium on Security and Privacy (SP)*, 393–409.

[51] Dan Wendlandt, David G. Andersen, and Adrian Perrig. 2008. Perspectives: Improving SSH-style host authentication with multi-path probing. In *Proceedings of the USENIX Annual Technical Conference*, 321–334.

A AGGREGATION LATENCY

Figure 9 shows a latency test for the P4-enabled NetFPGA target. There is a non-negligible difference between normal routing and STH aggregation which is less than 5 μs. Most importantly, however, we cannot observe any distinguisher for the different STH intervals as we aggregate.

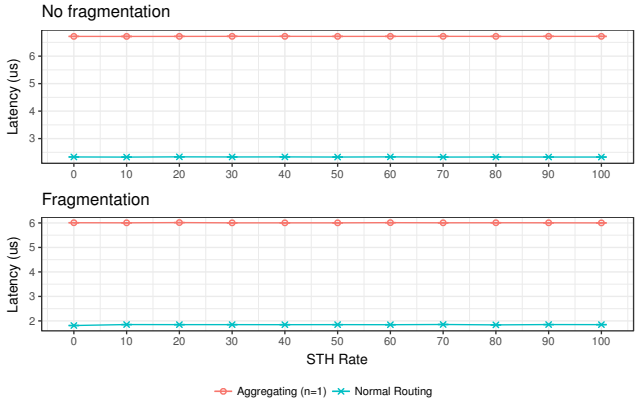


Figure 9: Latency as a function of match rate (P4-NetFPGA).